Final Report

Workload Forecast System - Group 3

Lappeenranta University of Technology

CT30A9300 Code Camp on Agile Java Development by Capgemini

Lappeenranta 30.10.2012

Poorang Vosough - 0392601

Negin Banaeianjahromi - 0392520

Nikolaos Paraschou – 0392575

Table of Contents

1 Scrum Method

Scrum is a framework which is used for agile software development. It is iterative and incremental. Despite the other frameworks, scrum does not predefine things to be done in details. In this framework, the way that things should be done depend on the software development team, because the team knows best how to solve the current problem. According to Mountain Goat Software [4], “Scrum relies on a self-organizing, cross-functional team.” Self-organizing means that there is no team leader who assigns particular tasks to a person or solves the problems individually. In scrum these issues are determined within the team. Cross-functionality of the team means that everyone is involved during the project, from planning to implementation.

1.1 Core Roles in Scrum Methodology

These roles are committed to the project and they represent the scrum team.

1.1.1 Product Owner

Represents the stakeholder and responsible for project’s success. The Product Owner leads the development effort by conveying his or her vision to the team, outlining work in the scrum backlog, and prioritizing it based on business value. Product Owner must also consider the stakeholders (to make sure their interests are included in the release) and the team (to make sure the release is developed by the deadline and within budget). As such, the Product Owner must be available to the team to answer questions and deliver direction [5].

1.1.2 Scrum Master

A person whose time is dedicated to ensuring a team’s ability to deliver on its sprint promises remains unobstructed. Scrum master is in charge of removing or eliminating all the impediments that make trouble for the team and prevent them to complete their task in the specific sprint.

1.1.3 Development Team

Development team is responsible for delivering a shippable product at the end of each sprint. Development Team consists of 3 to 9 people including: software engineers, architects, programmers, analysts, QA experts, testers, UI designers, etc. Scrum Team delivers products iteratively and incrementally at the end of each sprint.

1.2 Sprints

The main part of scrum is sprint, which lasts between one week and one month. Sprints have steady durations during the project. New sprint starts immediately after closing previous sprint. During each sprint, the team comes up with a shippable product for the customer.

1.2.1 Sprint Planning Meeting

In Scrum, every iteration begins with the sprint planning meeting. At this meeting, the Product Owner and the team negotiate which stories a team will tackle that sprint. Time-boxed according to eight hours, this meeting is a conversation between the Product Owner and the team. In this meeting the product owner decides which stories are of the highest priority to the release and which will generate the highest business value, but the team has the right to talk about their concerns or impediments.

The Sprint Planning Meeting consists of two parts. The two parts of the Sprint Planning Meeting answer the following questions [6]:

- What will be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

1.2.2 Daily Scrum

Each day during the sprint team has a 15-minute time-boxed event to synchronize activities and create a plan for the next 24 hours, which is called Daily Scrum or Daily Standup. During this meeting which takes place at the same place every day, each team member should answer to the following questions:

- What has been accomplished since the last meeting?
- What will be done before the next meeting?
- What obstacles are in the way?

1.2.3 Sprint Review

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. This informal meeting time-boxed to 4 hours during each sprint.

During this meeting Product Owner identifies what has been completed and what not. Also the team give a general overview of the whole sprint including what problems they ran into and how they solve them, team demonstrate the Done parts of the projects and answer the questions regarding the increment. In this point the team become ready for the next sprint by defining what to do next.

1.3 Scrum Artifacts

Artifacts defined by Scrum are specifically designed to maximize transparency of key information needed to ensure Scrum Teams are successful in delivering a "Done" Increment [6].

1.3.1 Product Backlog

Product Backlog is the most important artifact in Scrum. Product Backlog describes a to-do list of the customer priority based on the business value (User Stories). It is a list of everything that might be needed in the product and it is the only source of requirements during the project. The Product Owner is responsible for this artifact to provide content and priority. The Product Backlog is dynamic; it constantly changes to identify what the product needs to be appropriate, competitive, and useful. As long as a product exists, its Product Backlog also exists.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, and estimate. The Product Backlog is often ordered by value, risk, priority, and necessity. Top-ordered Product Backlog items drive immediate development activities. The higher the order, the more a Product Backlog item has been considered, and the more consensus exists regarding it and its value.

1.3.2 Sprint Backlog

This artifact provides a set of items from Product Backlog list for the next sprint. The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality. Picking tasks from Product Backlog list for the Sprint Backlog done by the Development team by asking "Can we also do this?".

The sprint backlog is the property of the Development Team, and all included estimates are provided by the Development Team. Often an accompanying task board is used to see and change the state of the tasks of the current sprint, like “to do”, “in progress” and “done”.

As new work is required, the Development Team adds it to the Sprint Backlog. As work is performed or completed, the estimated remaining work is updated. When elements of the plan are deemed unnecessary, they are removed. Only the Development Team can change its Sprint Backlog during a Sprint. The Sprint Backlog is a highly visible, real-time picture of the work that the Development Team plans to accomplish during the Sprint, and it belongs solely to the Development Team [4].

1.3.3 BurnDown Chart

The Burndown Chart is an artifact for demonstrating the remaining work versus time in the Sprint Backlog. It shows the daily progress of the team by updating everyday [5].

2 Program structure and features

Our application is built on top of Grails framework and it adheres to the MVC design pattern. The source code is divided in three distinct and interrelated parts:

The Model - where all the application logic is situated, the domain classes and the relationships between them are declared, and required constraints are defined.
The View - that consists of GSP files responsible to render and present the content to the user.
The Controller - which communicates with the model to fetch required data and forwards the data to the View in order to be presented.

In our implementation we created a basic model to represent certain business processes and logic and then we created and edited the controllers and views according to our plan. The result was an application that provides the following features:

Feature ID	Feature Description
F1	User Login: Consultants and Managers are identified. Each one is redirected to a different home page (home page for consultants, home page for managers).
F2	Easy navigation inside the application through a user friendly user interface.
F3	The consultant can create new forecasts at will, without approval from management.
F4	The consultant can enter all the necessary information about forecasts into the system. A forecast can have many activities associated with it. For each activity the consultant can enter the following information: activity type, client, comments, probability, COR/h, weeks that the consultant will be occupied on each activity, percentage of time that will be devoted on each activity per week. An activity can have many weeks associated with it.
F5	The forecasts entered by consultants are saved in the database and they can retrieved later on by both consultants and managers (for reviewing or editing).
F6	The managers can login to the system and read all the forecasts entered by all the consultants. They can edit the forecasts and make corrections and additions to them.
F7	The manager is able to create, read, update and delete consultants in the system.
F8	A director can log in to the system and be recognized as director.

Table 1: Implemented Application Features

3 First Sprint

During the first spring after we received the requirements from our customer and we identified the user stories based on them. After that we gave each user story a priority and the status. At this point we came up with the Product Backlog.

After finishing with Product Backlog we choose the highest priority tasks from Product Backlog and formed the Sprint Backlog; tasks to be implemented in the first Sprint. After that we started implementing the tasks which were in the Sprint Backlog and ended the first Sprint with a BurnDown chart.

3.1 Product Backlog

3.1.1 User Stories

User stories are one or more sentences which are written in Agile Software development for defining the functions that a system should provide for the customer. Generally a user story can be written as follows: As a <user> I want <some functionality> so that <some benefit is realized>.

Capgemini Company (one of the largest management consulting in the world) as our customer uses Microsoft Excel as its current forecasting workload which has its own challenges and does not provide a user friendly environment for the Capgemini´s employees to work in. Therefore we are supposed to provide a new workload forecast system for Capgemini Company which not only resolve Excel challenges but also provide a friendly environment its employees..

We began the process of developing the system with reading Capgemini´s requirement carefully and in detail. Then, from their requirements we identified different user stories for three kinds of Capgemini´s employees (consultant, manager and director).

Table 2 lists all the user stories that we added in the Product Backlog, along with their priorities.

Story ID	Story name	Status	Sprint	Priority
US1	As a consultant, I need to be able to log into the system, so that I can use it based on my authorization.	Done	1	1
US2	As a consultant, I need to have a user friendly UI, so that I can interact with the system effectively.	Done	1	1
US3	As a consultant, I want to create a new project without approval.	Done	1	1
US4	As a consultant, I want to be able to enter my workload forecasts into the system so that managers and directors can use them. A workload forecasts can contain information like activity type, client, project/comment, probe-%, etc.).	Done	1	1
US5	As a consultant, I want to be able to save the forecast that I entered, in a database, so that it can be retrieved in future.	Done	1	1
US6	As a Manager, firstly I want to login to the system and be recognized as a manager.	Done	1	1
US7	As a Manager, I want to be able to read and edit the workload forecasts of the consultants. My purpose is to review the consultants´ forecasts to and make corrections and additions to them.	Done	1	1
US8	As a Manager, I want to be able to edit (CRUD) users, user groups and teams.	30% Done	2	1
US9	As a Manager, I want to add users and assign them to proper teams.	40% Done	2	1
US10	As a Manager, I want to know exactly what the workload forecasts of my team's members are and all data related to those forecasts (activity type, client,	Planned	2	2

	project/comment, prob-%, etc.).			
US11	As a Manager, I want to see summaries and reports of workload forecasts. It is very important for me to know in real time what are the values of the following formulas: ARVE, URVE, COR.	Planned	2	2
US12	As a Manager, I want to see what the availability percentage of each consultant per week is.	Planned	2	2
US13	As a Manager, I want to see what the activity forecast is on a monthly basis (in percentage and in hours). Also, I want the estimated revenue (per activity per month) to be reported.	Planned	2	2
US14	As a Manager, I want the system to tell me "who did what and when". For example, if consultant X updated his workload forecast for activity Y, I want to know about it.	Planned	2	2
US15	As a Manager, I want to print reports.	Planned	2	2
US16	As a Director, firstly I want to login to the system and be recognized as a director, so that I can browse forecasting data and the summaries of them.	Done	1	1
US17	As a Director, I want to see what the activity forecast is on a monthly basis (in percentage and in hours). Also, I want the estimated revenue (per activity per month) to be reported.	Planned	2	2
US18	As a Director, I want to see the estimation of how much is left from total workload in percentage.	Planned	2	2
US19	As a Director, I want to be able to see the current forecast immediately after user changed data in database.	Planned	2	2
US20	As a Director, I want to be able to see ARVE	Planned	2	2

	(Assignment Rate Vacation Excluded), URVE (Utilization Rate Vacation Excluded) and COR (Charge-Out-Rate).			
US21	As a Director, I want to see the modification history.	Planned	2	2
US22	As a Director, I want to print reports.	Planned	2	2

Table 2: Product Backlog

3.2 Sprint Backlog

Here in Table 3, you can see the user stories which had priority 1, so we put them in the first sprint. In fact, these are the user stories that we have completed them in the first sprint.

Story ID	Story name	Status	Sprint	Priority
US1	As a consultant, I need to be able to log into the system, so that I can use it based on my authorization.	Done	1	1
US2	As a consultant, I need to have a user friendly UI, so that I can interact with the system effectively.	Done	1	1
US3	As a consultant, I want to create a new project without approval.	Done	1	1
US4	As a consultant, I want to be able to enter my workload forecasts into the system so that managers and directors can use them. A workload forecasts can contain information like activity type, client, project/comment, probe-%, etc.).	Done	1	1
US5	As a consultant, I want to be able to save the forecast that I entered, in a database, so that it can be retrieved in future.	Done	1	1
US6	As a Manager, firstly I want to login to the system and be recognized as a manager.	Done	1	1
US7	As a Manager, I want to be able to read and edit the workload forecasts of the consultants. My purpose is to review the consultants´ forecasts to and make corrections and additions to them.	Done	1	1
US8	As a Manager, I want to be able to edit (CRUD) users, user groups and teams.	30% Done	2	1
US9	As a Manager, I want to add users and assign them to proper teams.	40% Done	2	1
US16	As a Director, firstly I want to login to the system and be recognized as a director, so that I can	Done	1	1

	browse forecasting data and the summaries of them.			

Table 3: *Sprint Backlog*

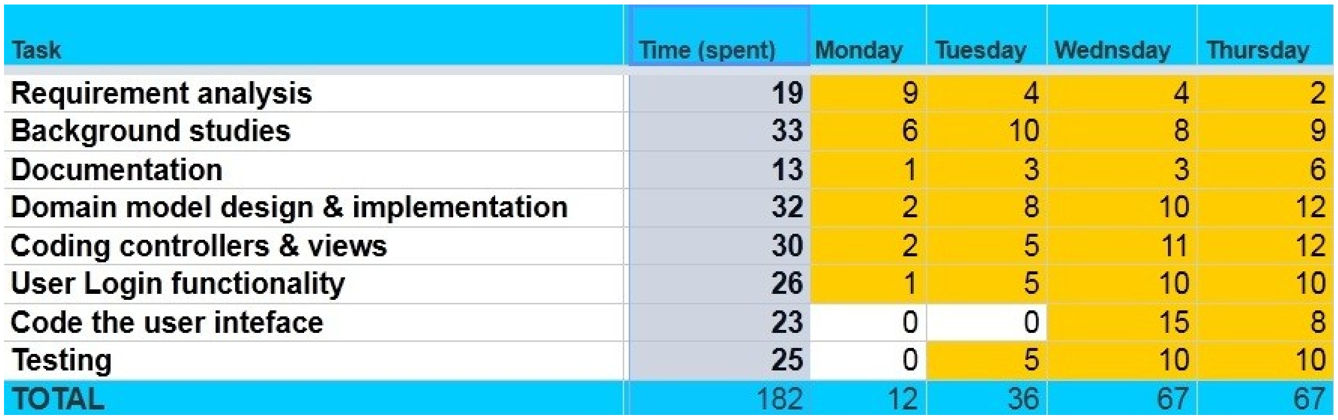
The following table shows the relation between user stories and implemented features (implemented features are listed in Table 1). What user stories led to what features:

User Story	Feature ID
US1	F1
US2	F2
US3	F3
US4	F4
US5	F5
US6	F1
US7	F6
US8	F7
US9	F7
US16	F8

Table 4: What user stories led to what features

3.3 BurnDown Chart

Figure 1 illustrates the time we spent as a team to each individual task of the first sprint.



Task	Time (spent)	Monday	Tuesday	Wednsday	Thursday
Requirement analysis	19	9	4	4	2
Background studies	33	6	10	8	9
Documentation	13	1	3	3	6
Domain model design & implementation	32	2	8	10	12
Coding controllers & views	30	2	5	11	12
User Login functionality	26	1	5	10	10
Code the user inteface	23	0	0	15	8
Testing	25	0	5	10	10
TOTAL	182	12	36	67	67

Figure 1: Time usage during the first sprint

In Burndown chart below which is based on Figure 1 statistics, the blue diagram is showing the time that we estimated for doing the tasks in sprint one. On the other hand, the red diagram is illustrating the realistic time usage spent for doing predefined tasks in sprint one. However, this chart is just showing the first sprint statistics, It means that time usage for next sprints will be much less than the first one. Basically, it is because of this fact that in Scrum methodology high priority features of the application are being done in the very first time of the development process and of course they are more time consuming.

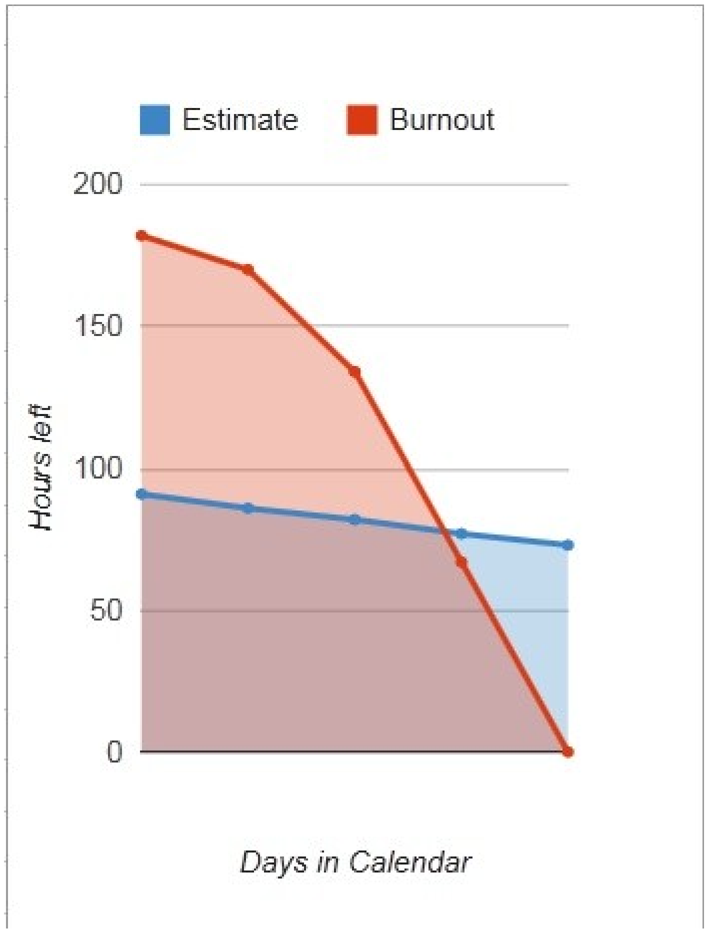


Figure 2: Burndown chart

4 Second Sprint

In the second sprint we completed the documentation and finalized the report.

5 MVC pattern with Grails

Grails is an open source web application framework that combines several technologies to allow rapid, dynamic and robust development on top of Java. The way in which it is architectured provides an out of the box mould of the MVC pattern. Right from the very beginning of a new web application project, all the necessary actions of establishing the foundation of MVC pattern are performed automatically by the framework. More specifically, a directory structure of the source code is created that separates the model, the view and the controller source files.

5.1 Model

In the model subdirectory of the source code, the developer should create the Domain Classes. Those are classes that represent the domain entities and the relationships between them. Grails will automatically create the necessary database schemas based on the model structure. Domain classes consist the core to any business application produced by Grails. Business processes and business logic are implemented by domain classes. The relationships that can be formed between domain entities are one-to-one, one-to-many, or many-to-many.

All the relationships and constraints defined by the programmer in the domain classes are handled automatically by the framework when creating the database. The programmer doesn't have to worry about writing SQL code or interfering with the underlying database management system. This provides great flexibility and the opportunity to do rapid application development since the developer is completely decoupled from the database. The framework allows the developer to operate only on the domain level.

The way in which Grails performs the aforementioned mapping between the domain model and the database is through the GORM (Grails Object Relational Mapping) implementation. The technology used under the hood is Hibernate 3 and the domain classes are written in Groovy (they can also be written directly in Java) [1].

In our implementation we created the following domain classes:

- **User.groovy** - Implements a user. Contains information like real name, login username, login password, etc.
- **UserRole.groovy** - Implements a user role which can take the values Consultant, Manager and Director. The user role is associated with the user in a one-to-one relationship.
- **Forecast.groovy** - Implements a Forecast. A forecast is a workload forecast that each consultant should create. In our implementation the user is associated with forecast with a many-to-many

relationship, which means that a user can have many forecasts (perhaps a forecast for each month, or it could be as the consultant wants). A forecast can have many activities.

- **Activity.groovy** - Implements an Activity. An activity is a certain obligation/task that a consultant has to fulfil and he should make a prediction about it. The information included in an activity contains the type of the activity, the client where it will take place, some comments, the probability that the activity will actually happen, the amount of money that will be earned per hour during the activity and finally the weeks (specific periods of time) that will be occupied with this activity. As already mentioned, the forecast is associated with the activity with a one-to-many relationship, which means that a forecast can have many activities.
- **Week.groovy** - Implements a specific period of time. Weeks can be created by the consultants depending on their schedule. A week has a start date and an end date which specifies it. Actually it can be any period of time (not necessarily a week, it is called week by convention). The week also contains the percentage of time (occupation), that is how occupied will the consultant be during this week on the activity that is associated with the week. The activity is associated with the week in a one-to-many relationship, which means that an activity can have many weeks associated with it.
- **Client.groovy** - Implements a client. This domain class is used to represent and allow the creation of clients. By client we mean clients of the company that operates the workload forecast system. The consultants are offering their services to these clients. The information included for each client includes just the name of the client. The activity is associated with the client in a one-to-one relationship, which means that each activity has only one client.
- **Login.groovy** - Implements login functionality. This domain class is used to provide login functionality. It doesn't offer real domain value. It is actually a helper class that allows the creation of the LoginController and login views which are necessary to implement the login feature.

The actual database schema created automatically by Grails after the processing of our domain model is depicted in Figure 3: Database Model.

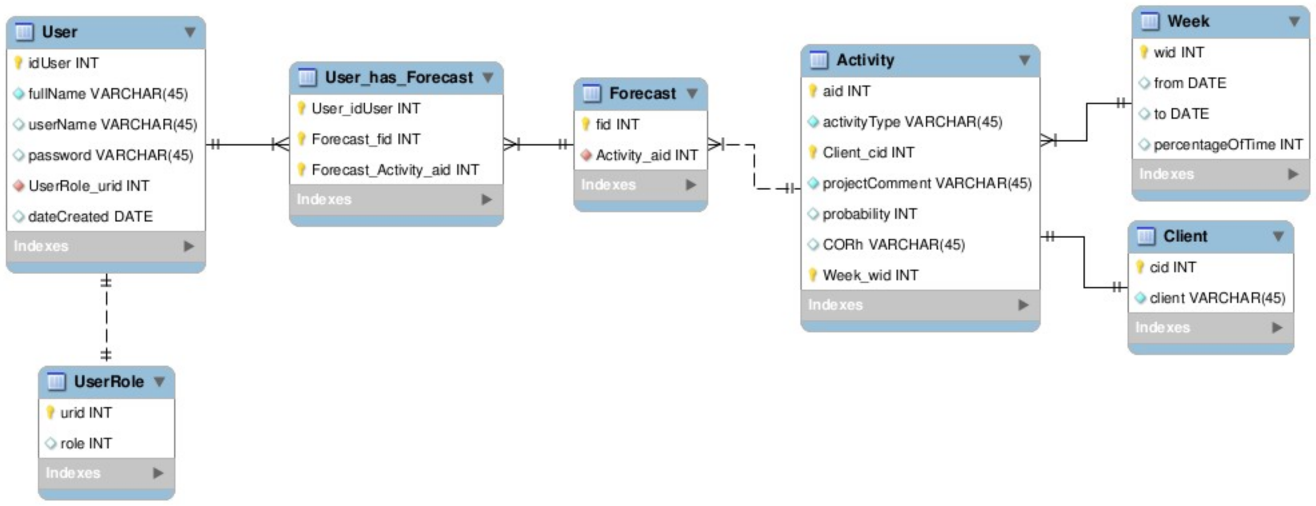


Figure 3: Database Model

5.2 Controller

The Controllers are responsible for handling user requests and preparing responses [2]. The response can be created directly from the controller or it can be forwarded to a view which will prepare and present it. Controllers in Grails are named after the domain classes which they control adding the Controller word after the name of the domain class (e.g. the controller of the User.groovy should be UserController.groovy).

Grails comes with a feature called scaffolding. With scaffolding activated in a domain class, the framework will automatically generate the appropriate CRUD (Create, Read, Update, Delete) controllers for that class. In addition to that, the necessary views will also be created automatically [3].

In our implementation we have activated scaffolding in all domain classes only during the first build. When the needed controllers and views were created automatically by Grails, we deactivated scaffolding and proceeded by making the desired corrections and changes to the controllers and views.

Regarding the controllers, we made the necessary modifications to implement the login functionality. In all the controller classes we added the auth() method which verifies if the user requesting the content offered by the controller is authorized to receive it (logged in or not). Furthermore, we added the beforeInterceptor member which is responsible to redirect all user requests to the auth() method before serving the request (every time the user asks something from the controller he must first be authorized).

Following is a list of all the implemented controllers (one controller for each domain class):

- **UserController.groovy** - Implements CRUD functionality for users, provides authorization control.
- **UserRoleController.groovy** - Implements CRUD functionality for user roles, provides authorization control.
- **ForecastController.groovy** - Implements CRUD functionality for forecasts, provides authorization control.
- **ActivityController.groovy** - Implements CRUD functionality for activities, provides authorization control.
- **WeekController.groovy** - Implements CRUD functionality for weeks, provides authorization control.
- **ClientController.groovy** - Implements CRUD functionality for clients, provides authorization control.
- **LoginController.groovy** - Implements the login functionality. Checks if the user is logged in. If not, it redirects the user to the login page. If yes, it checks the user's role and according to the user's role it redirects him to the proper page (consultant home, manager home). Also performs the user authentication/login (authenticate() method) and the user logout (logout() method).

5.3 View

The view technology in Grails is called Groovy Server Pages (GSP) [2]. GSP resembles similar technologies like ASP and JSP and it is based on markup (XHTML) and GSP tags to render and present content to users.

As with the controllers, scaffolding also produces automatically the views for the CRUD functionality. We used the automatically generated views for most of the domain classes but we made modifications to them in order to suit our needs.

More specifically, we modified the views accordingly to match with the client's look and feel preferences by including the client's logo, we made modifications to the CSS in several places to include a customized menu bar, and several other changes to improve the user experience while using the application. Moreover, we added a new custom view which provides the login page and it is connected with the LoginController. Last but not least, we created appropriate views for the home pages of consultants and managers to which they are redirected after successful login (consultants and managers do not have the same homepages because they have different rights).

6 The development process and lessons learned

6.1 Development process

In this Code Camp, we had almost one week for development process, 15th to 19th of October 2012. In the beginning of the process on Monday 15th, we defined user stories based on the requirement which was received from the customer. The next day on 16th, we prioritized the user stories based on their importance level in our point of view. Later on, based on defined priority levels for user stories we put them in two different sprints. In this case, we formed the product backlog and started to work on implementation phase of the project. In implementation phase on 17th and 18th, we started learning Grails and programming at the same time. For saving the time, we divided the tasks related to Model and View of the application between our members, so we could manage to develop different parts of the project in parallel; in this case, we could finish all of the tasks were defined in sprint one before deadline.

6.2 Methodology and tools

In this Code Camp we were supposed to develop our application based on Scrum methodology and using Grails web framework. By applying Scrum methodology in our application we learned how we can develop an application in a short period of time based on Agile development methods. In fact, we learned how we are supposed to prioritize user stories of the application into different sprints based on their importance level and also based the strict deadline of the project.

Moreover, by using Grails we learned how MVC pattern can be effective in programming. It provides for the programmers the possibility of working on different parts of the application in parallel. One team member can work on the logic of the application (model) while the other members can work on the user interface of the application (view) at the same time.

6.3 Lectures

In this Code Camp we were attending in different lectures which tried to make a better understanding for the participants of what is really demanded in this course. There was a lecture for introducing the company which helped the teams to know Capgemini and what they need, on the other hand, other lectures about Agile development, Scrum methodology and Grails gave the teams the start point of researching about the concepts that they are supposed to use in their application. Basically, lectures helped the teams to find the correct direction, although after that teams were supposed to get to destination by themselves.

6.4 Team working

In the beginning of the Code Camp we did not know concepts such as Scrum methodology, Agile development and Grails. On the other hand, we were supposed to learn them in a short period of time so we can use them for developing our application. Generally, learning and working at the same time increase stress on the team and affect team performance negatively. However, we learned from team working how we can learn together and use what we learned in the same time in our mutual project. In other words, cooperating as a team gave us the power of overcoming the difficulties and lead us to achieve our goals.

6.5 Working with other teams

Working with other teams in the same place was an interesting experience for us. Majority of the team members were not familiar with the concepts such as Scrum and Grails, so all teams had the same problem to solve. In this case, they could help each other and share what they learned. On the other hand, working in the same place with other teams increased the feeling of competition between, so each team tried its best to be the best one.

6.6 Presentation

In the last session of the Code Camp, teams presented what they developed in almost one week. From the team presentations we learned how different items can affect the final result. In fact, for being the winner team not only you must develop the best application with efficient logic and acceptable UI but also you should present it in a way that makes the attendance interested in what you developed and convince the you made the best application.

REFERENCES

[1] Object Relational Mapping (GORM) - Reference Documentation, retrieved from http://grails.org/doc/latest/guide/GORM.html , 27.10.2012

[2] The Web Layer - Reference Documentation, retrieved from http://grails.org/doc/latest/guide/theWebLayer.html , 27.10.2012

[3] Scaffolding - Reference Documentation, retrieved from http://grails.org/doc/latest/guide/scaffolding.html , 27.10.2012

[4] What is Scrum?-Reference Documentation, retrieved from http://www.mountaingoatsoftware.com/topics/scrum , 28.10.2012

[5] Scrum Product Owner, retrieved from http://scrummethodology.com/scrum-product-owner/ , 30.10.2012

[6] K.Schwaber and J.Sutherland. (October 2011). The Scrum Guide [Online]. Available: http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum_Guide.pdf , 30.10.2012